The DOE ACTS Collection

How can it be used?

UC Berkeley - CS267 - April 13, 2005

Tony Drummond

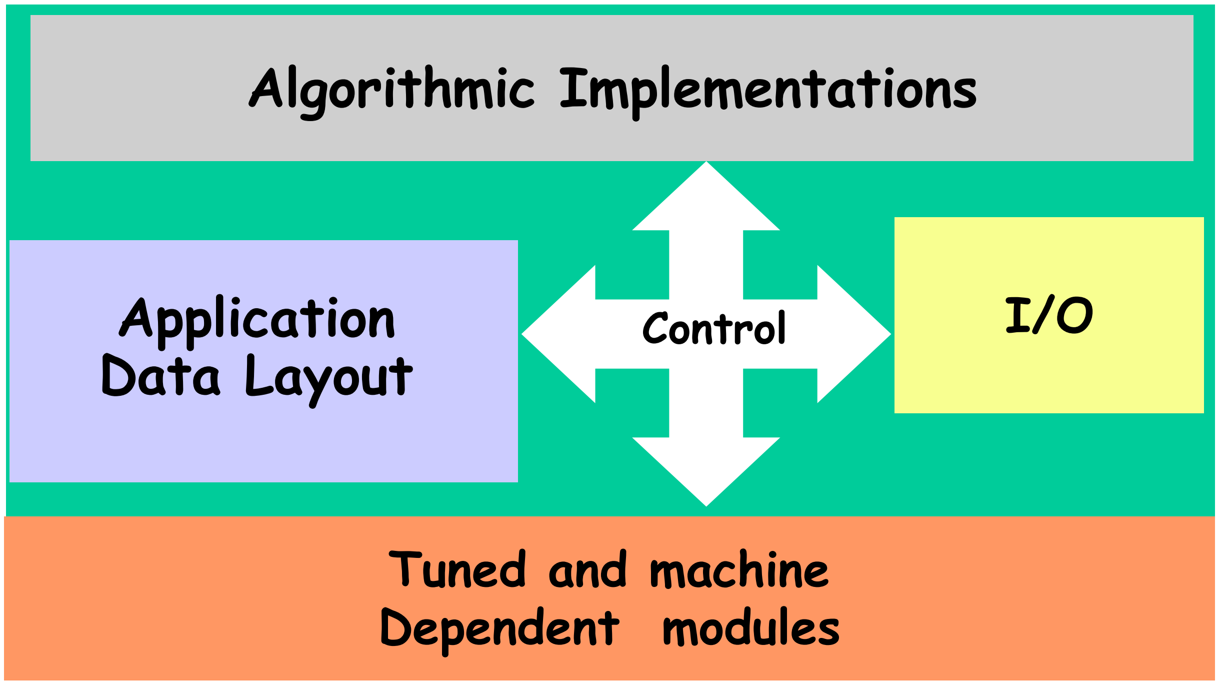
Lawrence Berkeley National Laboratory

acts-support@nersc.gov

ladrummond@lbl.gov

Motivation- Why do we need software libraries?

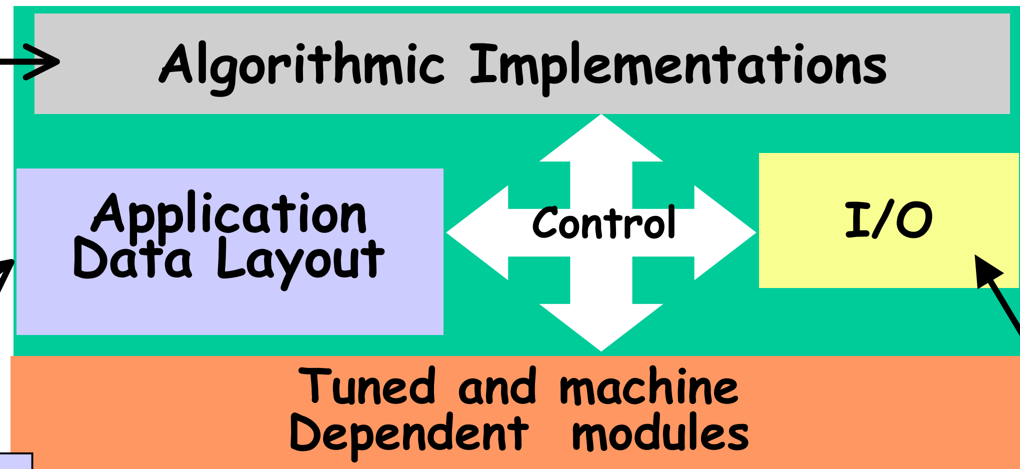
Large Scientific Codes:
A Common Programming Practice



Motivation- Why do we need software libraries?

New Architecture:
- May or may not need re-rewriting

New Developments:
- Difficult to compare



New Architecture:
- Extensive re-rewriting

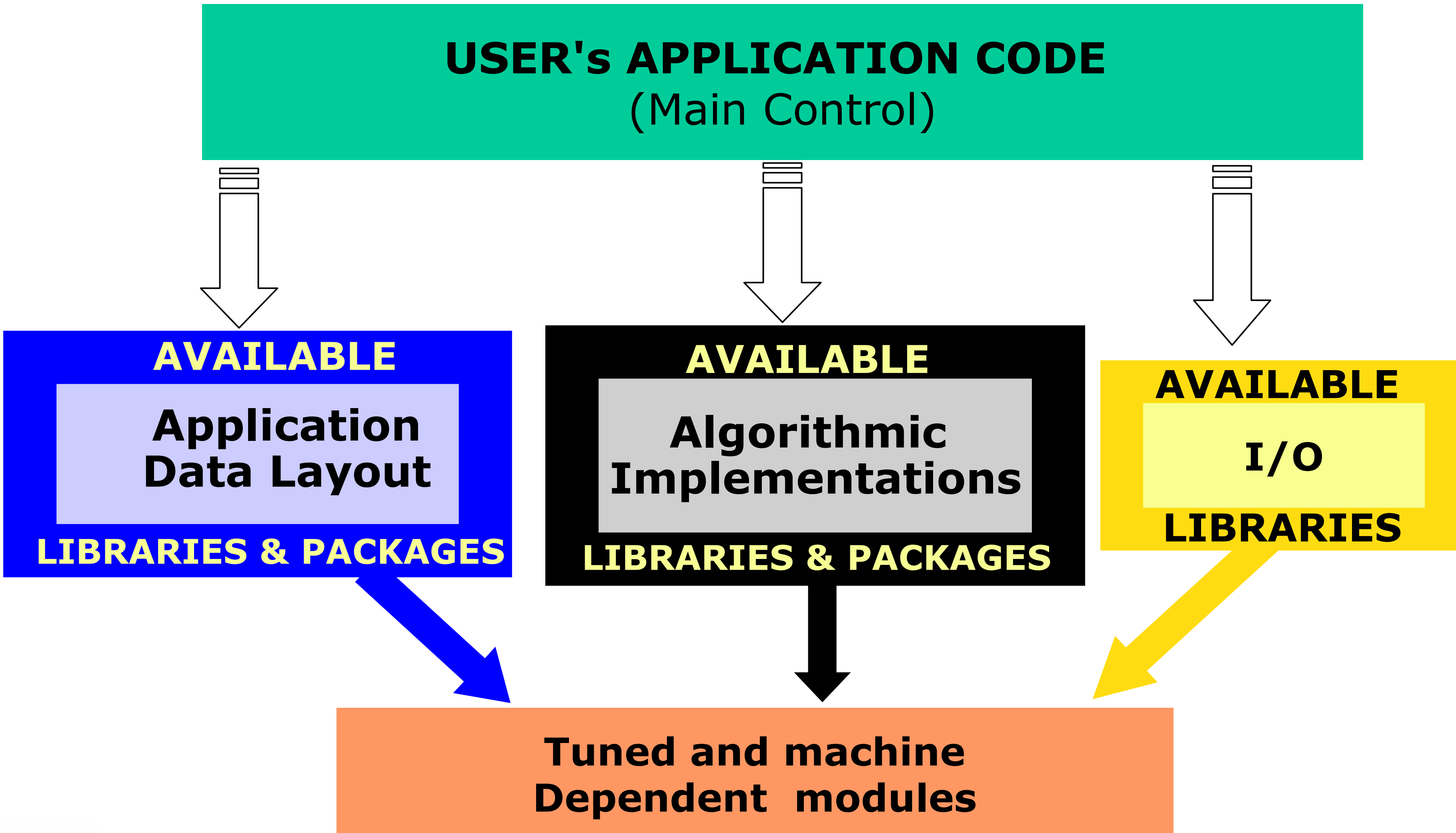
New or extended Physics:
- Extensive re-rewriting or increase overhead

New Architecture:
- Minimal to Extensive rewriting

New Architecture or S/W
- Extensive tuning
- May require new programming paradigms
- **Difficult to maintain!**

Motivation- Why do we need software libraries?

<u>An Alternative Approach</u>



Shortcomings?

"We need to move away from a coding style suited for serial machines, where every macrostep of an algorithm needs to be thought about and explicitly coded, to a higher-level style, where the compiler and library tools take care of the details. And the remarkable thing is, if we adopt this higher-level approach right now, even on today's machines, we will see immediate benefits in our productivity."

W. H. Press and S. A. Teukolsky, 1997
Numerical Recipes: Does This Paradigm Have a future?

ACTS Tools

Category	Tool	Functionalities
Numerical $Ax = b$ $Az = \lambda z$ $A = U \Sigma V^T$ PDEs ODEs	**AztecOO**	Algorithms for the iterative solution of large sparse linear systems.
	Hypre	Algorithms for the iterative solution of large sparse linear systems, intuitive grid-centric interfaces, and dynamic configuration of parameters.
	PETSc	Tools for the solution of PDEs that require solving large-scale, sparse linear and nonlinear systems of equations.
	OPT++	Object-oriented nonlinear optimization package.
	SUNDIALS	Solvers for the solution of systems of ordinary differential equations, nonlinear algebraic equations, and differential-algebraic equations.
	ScaLAPACK	Library of high performance dense linear algebra routines for distributed-memory message-passing.
	SuperLU	General-purpose library for the direct solution of large, sparse, nonsymmetric systems of linear equations.
	TAO	Large-scale optimization software, including nonlinear least squares, unconstrained minimization, bound constrained optimization, and general nonlinear optimization.
Code Development	**Global Arrays**	Library for writing parallel programs that use large arrays distributed across processing nodes and that offers a shared-memory view of distributed arrays.
	Overture	Object-Oriented tools for solving computational fluid dynamics and combustion problems in complex geometries.
Code Execution	**CUMULVS**	Framework that enables programmers to incorporate fault-tolerance, interactive visualization and computational steering into existing parallel programs
	Globus	Services for the creation of computational Grids and tools with which applications can be developed to access the Grid.
	TAU	Set of tools for analyzing the performance of C, C++, Fortran and Java programs.
Library Development	**ATLAS**	Tools for the automatic generation of optimized numerical software for modern computer architectures and compilers.

Software Selection

```
CALL BLACS_GET( -1, 0, ICTXT )
 CALL BLACS_GRIDINIT( ICTXT, 'Row-major', NPROW, NPCOL )
:
 CALL BLACS_GRIDINFO(ICTXT, NPROW, NPCOL, MYROW, MYCOL)
:
:
 CALL PDGESV( N, NRHS, A, IA, JA, DESCA, IPIV, B, IB, JB, DESCB,
 $              INFO )
```

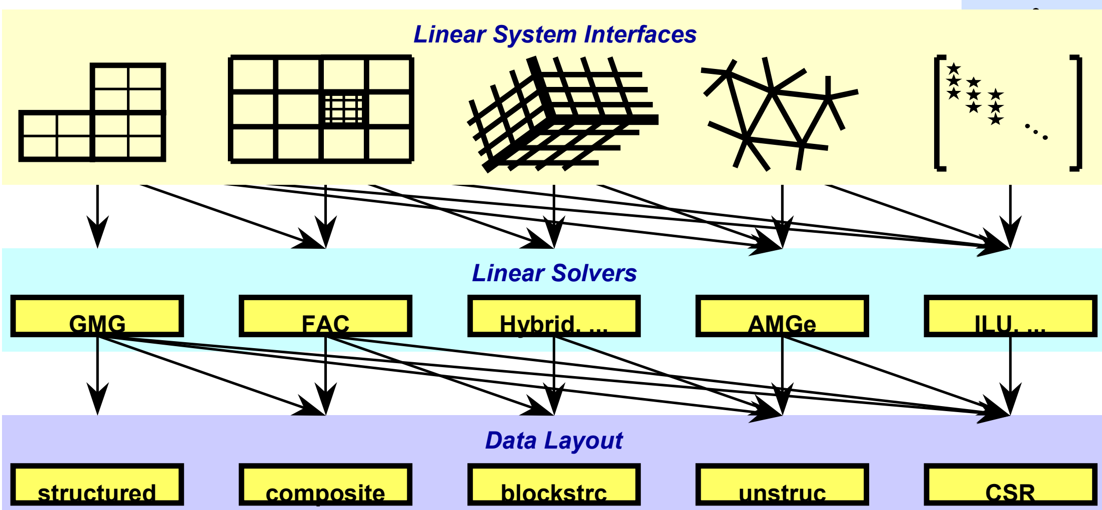
Language Calls

Command lines

- -ksp_type [cg,gmres,bcgs,tfqmr,…]
- -pc_type [lu,ilu,jacobi,sor,asm,…]

More advanced:

- -ksp_max_it <max_iters>
- -ksp_gmres_restart <restart>
- -pc_asm_overlap <overlap>
- -pc_asm_type [basic,restrict,interpolate,none]



Problem Domain

ACTS Numerical Tools: *Functionality*

Computational Problem	Methodology	Algorithms	Library
Linear Equations Problems	Direct	LU factorization	ScaLAPACK (dense) SuperLU (sparse)
		Cholesky factorization	ScaLAPACK
		LDL^T factorization (tridiagonal matrices)	ScaLAPACK
		QR factorization	ScaLAPACK
		QR with Column Pivoting factorization	ScaLAPACK
		LQ factorization	ScaLAPACK
		Complete Orthogonal factorization	ScaLAPACK
		Generalized QR factorization	ScaLAPACK • General QR, LQ,QL, RQ and RZ

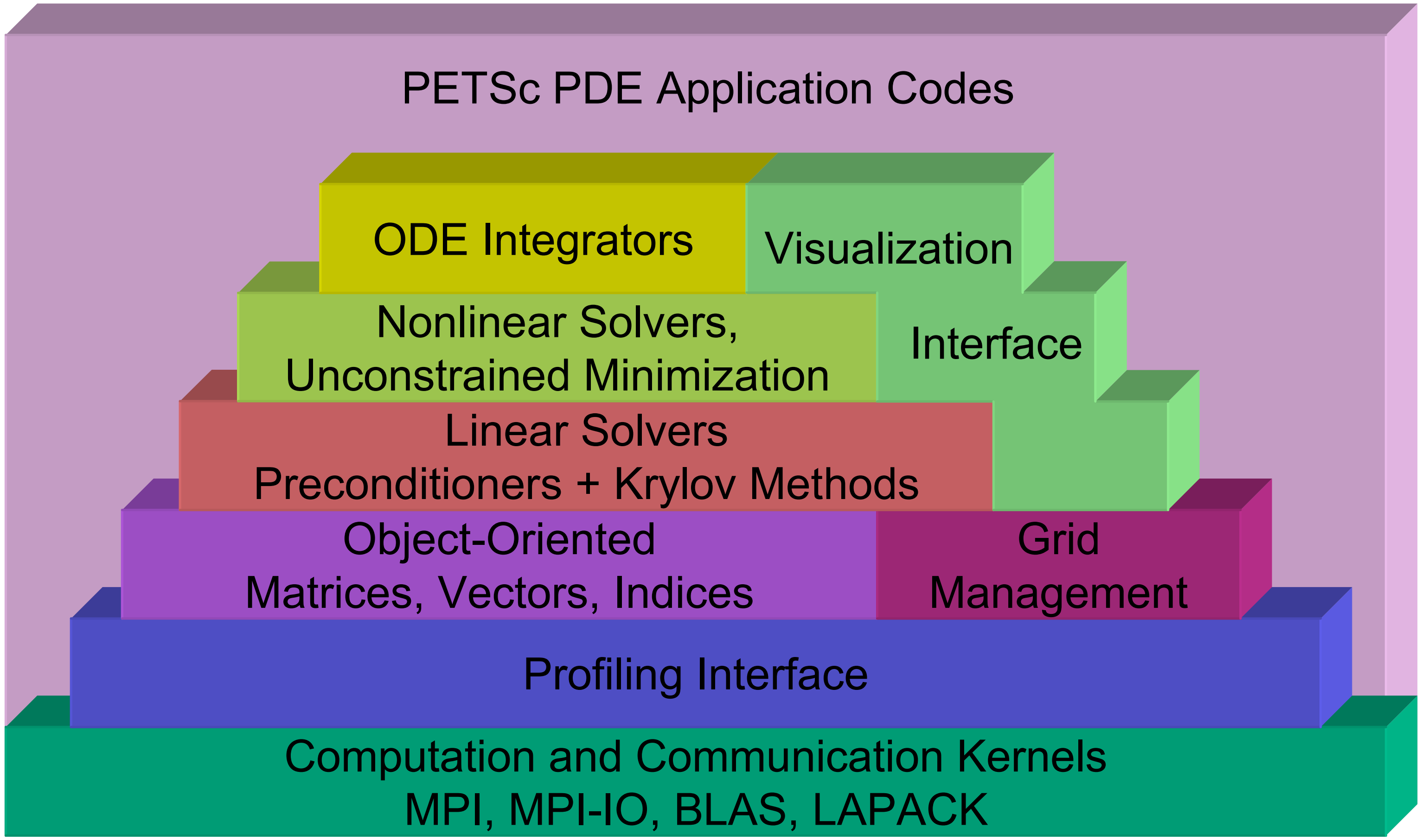
ACTS Numerical Tools: *Functionality*

Computational Problem	Methodology	Algorithms	Library
Linear Equations Problems	Iterative	Conjugate Gradient (CG)	AztecOO (Trilinos) PETSc
		GMRES	AztecOO Hypre PETSc
		CGS (CG Squared)	AztecOO PETSc
		Bi-CG-Stab	AztecOO PETSc
		Quasi-Minimal Residual (QMR)	AztecOO
		Transpose Free QMR	AztecOO PETSc
		SYMMLQ (symmetric LQ)	PETSc
		Preconditioned CG	AztecOO Hypre PETSc
		Richardson	PETSc
		Block Jacobi preconditioner	AztecOO Hypre PETSc
		Point Jacobi preconditioner	AztecOO
		Least-squares polynomials	AztecOO
		SOR preconditioner	PETSc

ACTS Numerical Tools: *Functionality*

Computational Problem	Methodology	Algorithms	Library
Linear Equations Problems	iterative (cont.)	Overlapping Additive Schwarz (ASM) preconditioner	PETSc
		Approximate Inverse	Hypre
		Sparse LU preconditioner	AztecOO Hypre PETSc
		Incomplete LU (ILU) preconditioner	AztecOO Hypre PETSc
	Multigrid (MG)	MG preconditioner	Hypre PETSc
		Algebraic Multigrid	ML (Trilinos) Hypre
		Semicoarsening	Hypre

Structure of PETSc



PETSc Numerical Components

Nonlinear Solvers

Newton-based Methods		Other
Line Search	Trust Region	

Time Steppers

Euler	Backward Euler	Pseudo Time Stepping	Other

Krylov Subspace Methods

GMRES	CG	CGS	Bi-CG-STAB	TFQMR	Richardson	Chebychev	Other

Preconditioners

Additive Schwartz	Block Jacobi	Jacobi	ILU	ICC	LU (Sequential only)	Others

Matrices

Compressed Sparse Row (AIJ)	Blocked Compressed Sparse Row (BAIJ)	Block Diagonal (BDIAG)	Dense	Matrix-free	Other

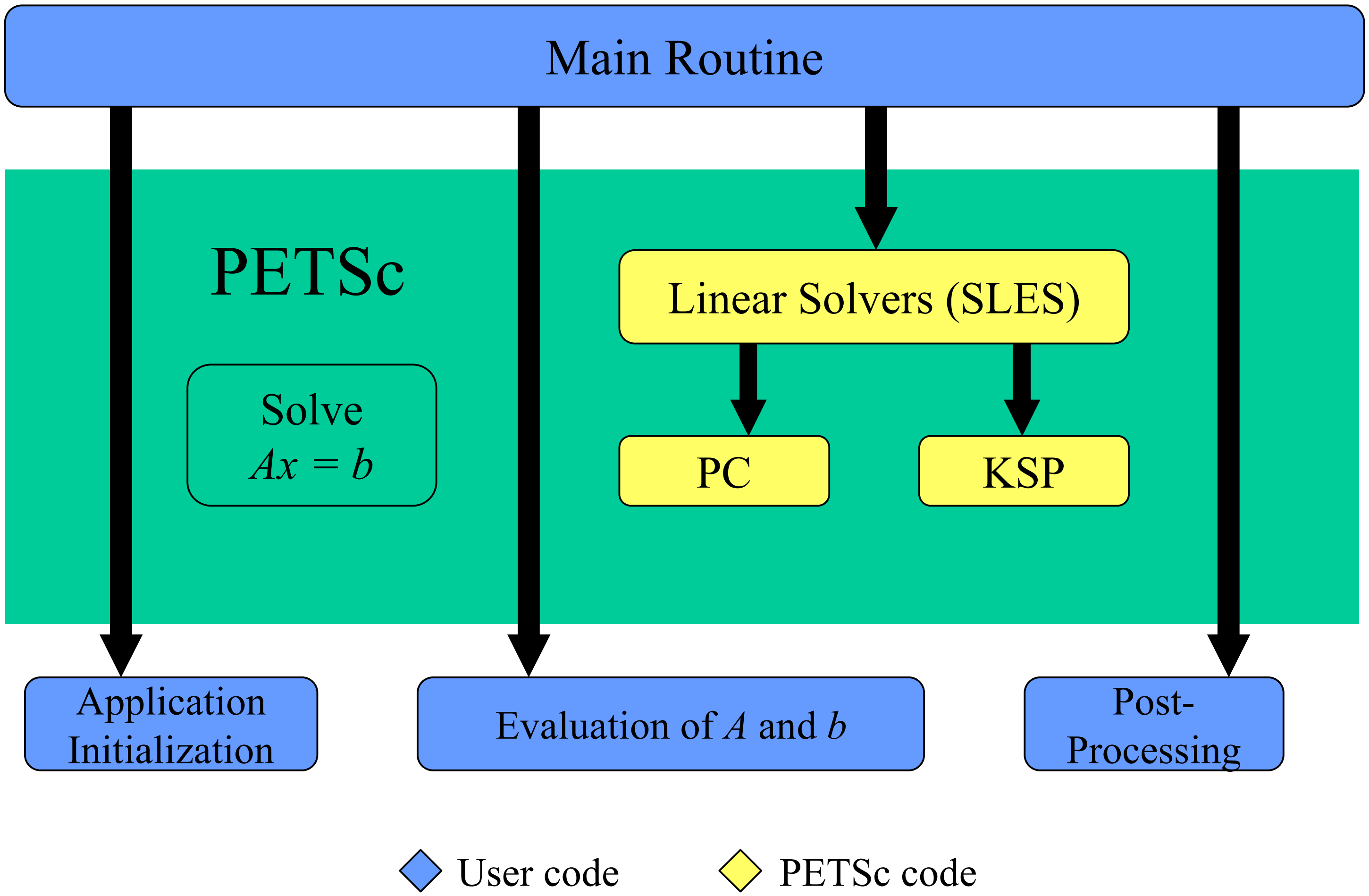
Distributed Arrays

Index Sets

Indices	Block Indices	Stride	Other

Vectors

PETSc Krylov Subspace Methods



Partial list of Linear Solvers in PETSc

Krylov Methods (KSP)

- Conjugate Gradient
- GMRES
- CG-Squared
- Bi-CG-stab
- Transpose-free QMR
- etc.

Preconditioners (PC)

- Block Jacobi
- Overlapping Additive Schwarz
- ICC, ILU via BlockSolve95
- ILU(k), LU (sequential only)
- etc.

PETSc Example - Basic Linear Solver in C

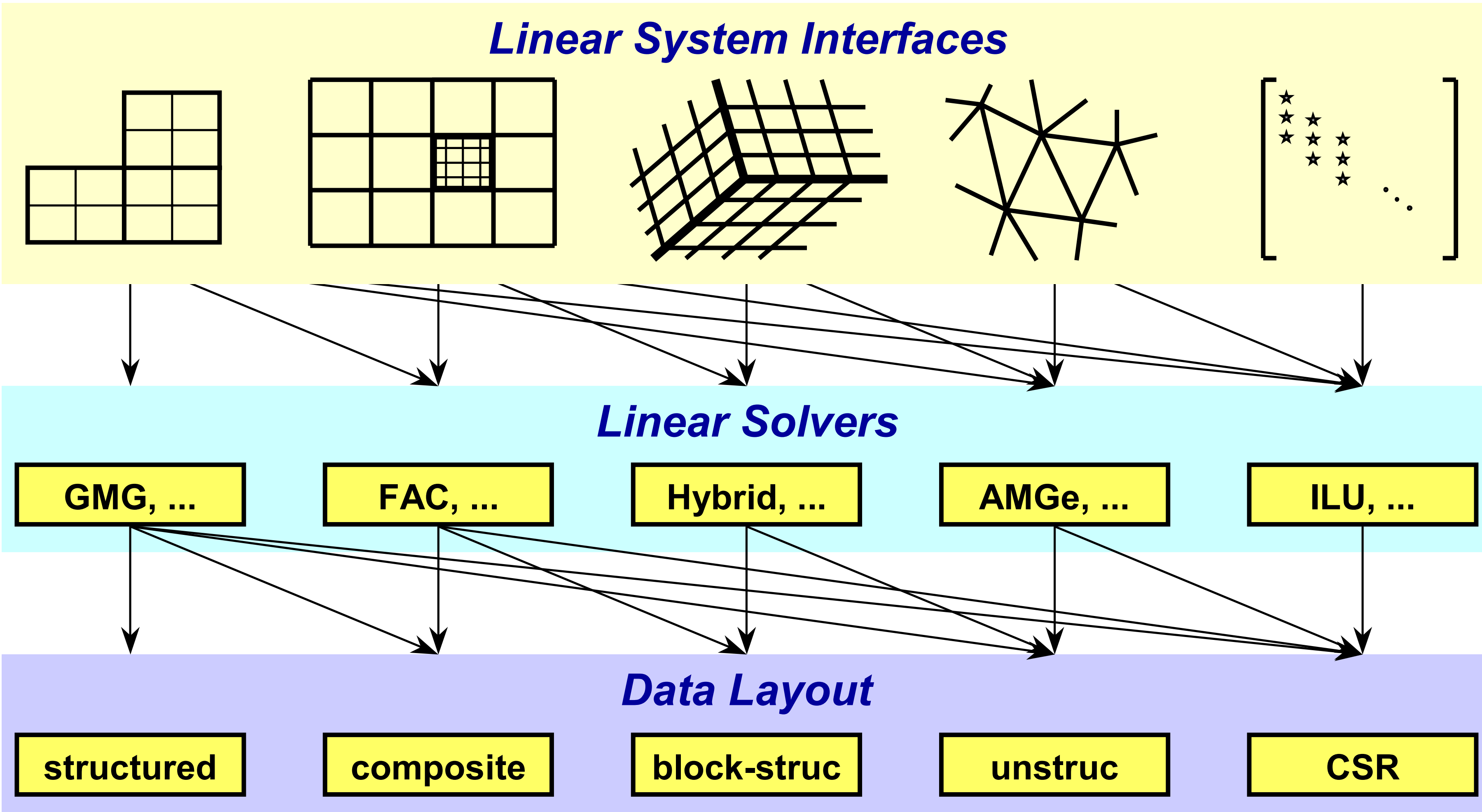
```
SLES   sles;          /*  linear solver context  */
Mat    A;             /*  matrix  */
Vec    x, b;          /*  solution, RHS vectors  */
int    n, its;        /*  problem dimension, number of iterations  */

MatCreate(MPI_COMM_WORLD,PETSC_DECIDE,PETSC_DECIDE,
          n,n,&A);                          /* assemble matrix */
VecCreate(MPI_COMM_WORLD,PETSC_DECIDE,n,&x);
VecDuplicate(x,&b);                         /* assemble RHS vector */

SLESCreate(MPI_COMM_WORLD,&sles);
SLESSetOperators(sles,A,A,DIFFERENT_NONZERO_PATTERN);
SLESSetFromOptions(sles);
SLESSolve(sles,b,x,&its);
SLESDestroy(sles);
```

Original Matrix from Linear System

Preconditioning matrix

SAME_NON_ZERO_PATTERN
SAME_PRECONDITIONER

Hypre Conceptual Interfaces



Hypre Conceptual Interfaces

- **Structured-Grid Interface (Struct)**
  - *applications with logically rectangular grids*
- **Semi-Structured-Grid Interface (SStruct)**
  - *applications with grids that are mostly—but not entirely—structured (e.g., block-structured, structured AMR, overset)*
- **Finite Element Interface (FEI)**
  - *unstructured-grid, finite element applications*
- **Linear-Algebraic Interface (IJ)**
  - *applications with sparse linear systems*

Hypre Conceptual Interfaces

- Before writing your code:
  - choose a conceptual interface
  - choose a solver / preconditioner
  - choose a matrix type that is compatible with your solver / preconditioner and conceptual interface

- Code Development
  - build auxiliary structures (e.g., grids, stencils)
  - build matrix/vector through conceptual interface
  - build solver/preconditioner
  - solve the system
  - get desired information from the solver

Hypre's IJ interface:
setting up the Solver

List of Solvers and Preconditioners per Conceptual Interface

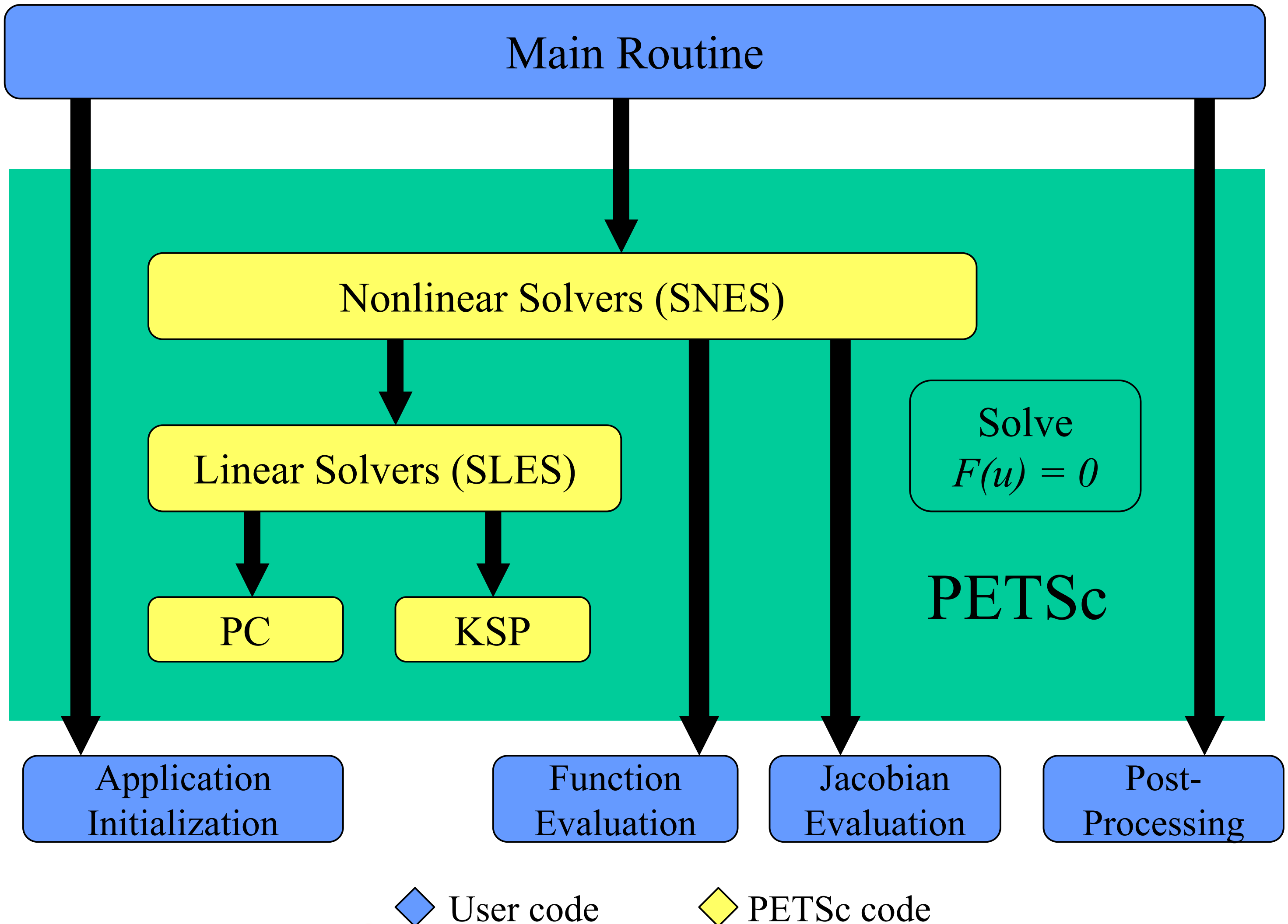
Solvers	System Interfaces			
	Struct	SStruct	FEI	IJ
Jacobi	X			
SMG	X			
PFMG	X			
BoomerAMG	X	X	X	X
ParaSails	X	X	X	X
PILUT	X	X	X	X
Euclid	X	X	X	X
PCG	X	X	X	X
GMRES	X	X	X	X

ACTS Numerical Tools: *Functionality*

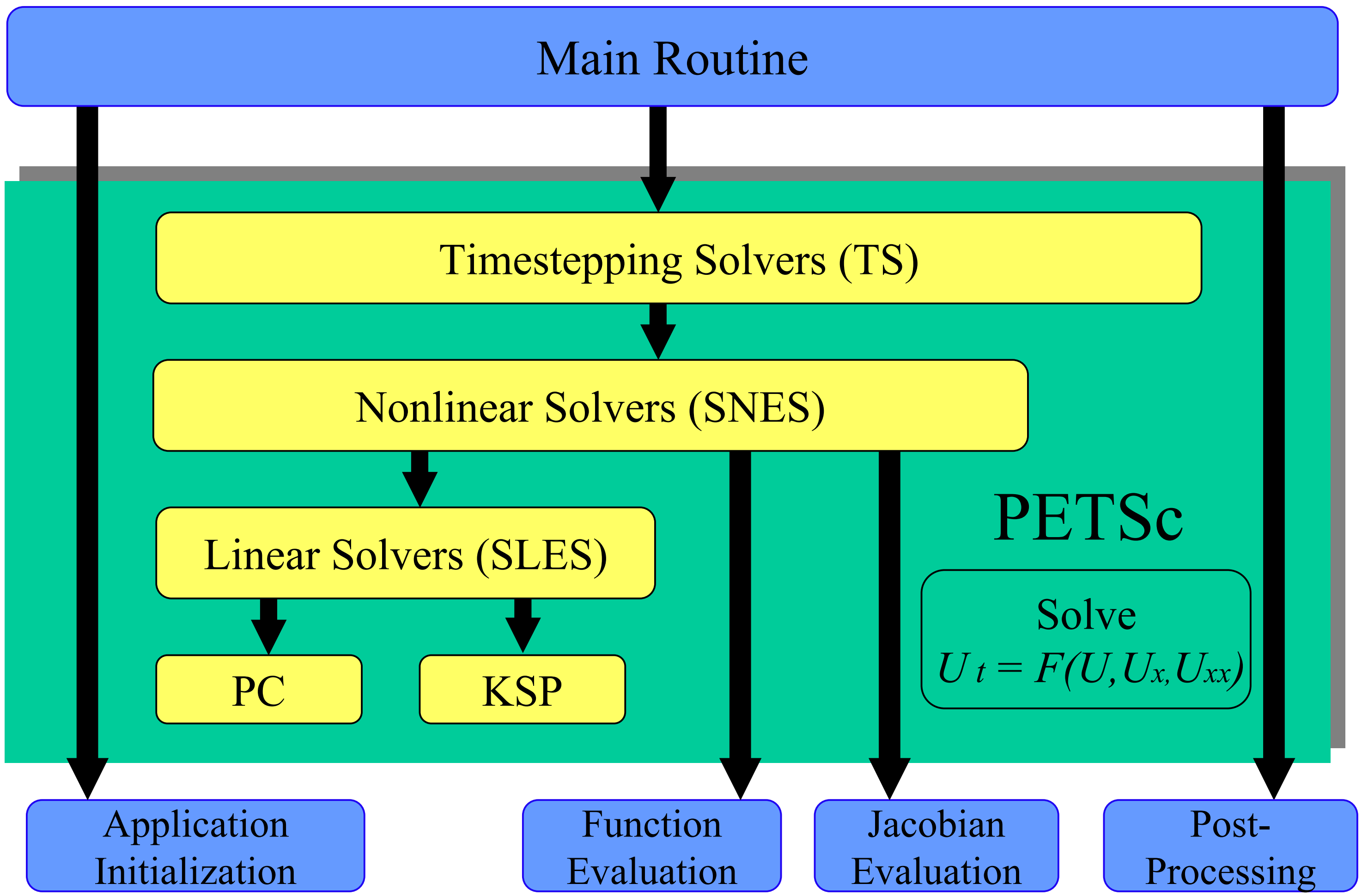
Computational Problem	Methodology	Algorithms	Library
Linear Least Squares Problems	Least squares solution	$minimize_x \|\|b - Ax\|\|_2$	ScaLAPACK
	Minimum norm solution	$minimize_x \|\|x\|\|_2$	ScaLAPACK
	Minimum norm least squares	$minimize_x \|\|x\|\|_2$ and $\|\|b - Ax\|\|_2$	ScaLAPACK
Standard Eigenvalue Problem	Symmetric Eigenvalue Problems	$Az = \lambda z$ for $A = A^T$ or $A = A^H$	ScaLAPACK (dense) SLEPc (sparse)
Singular Value Problems	Singular Value Decomposition	$A = U \Sigma V^T$ $A = U \Sigma V^H$	ScaLAPACK(dense) SLEPc (sparse)
Generalized Symmetric Definite Eigenproblem	Eigenproblem	$Az = \lambda Bz$, $ABz = \lambda z$, $BAz = \lambda z$	ScaLAPACK (dense) SLEPc (sparse)
Non-linear Equations Problems	Newton-based	Line Search	PETSc
		Trust Regions	PETSc
		Pseudo-transient continuation	PETSc
		Matrix free	PETSc
Non-linear Optimization Problems	Newton-based	Newton	OPT++ TAO
		Finite Difference Newton	OPT++
		Quasi Newton	OPT++ TAO (LMVM)
		Nonlinear Interior Point	OPT++ TAO

BERKELEY LAB

PETSc Nonlinear Solver (SNES)



Time Dependent PDE Solution



ACTS Numerical Tools: *Functionality*

Computational Problem	Methodology	Algorithms	Library
Non-linear Optimization Problems (cont.)	CG	Standard nonlinear CG	OPT++ TAO
		Limited memory BFGS	OPT++ (unconstrained)
		Gradient Projection	TAO
	Direct Search	Without derivative Information	OPT++
	Semismooth	Infeasible semismooth	TAO
		Feasible semismooth	TAO
Ordinary differential equations	Integration	Variable coefficient forms of the Adams-Moulton	CVODE (SUNDIALS)
	Backward differential formula	Direct and iterative solvers	CVODE (SUNDIALS)
Nonlinear Algebraic Equations	Inexact Newton	Line search	KINSOL (SUNDIALS)
Differential-Algebraic Equations	Backward differential formula	Direct and iterative solvers	IDA (SUNDIALS)

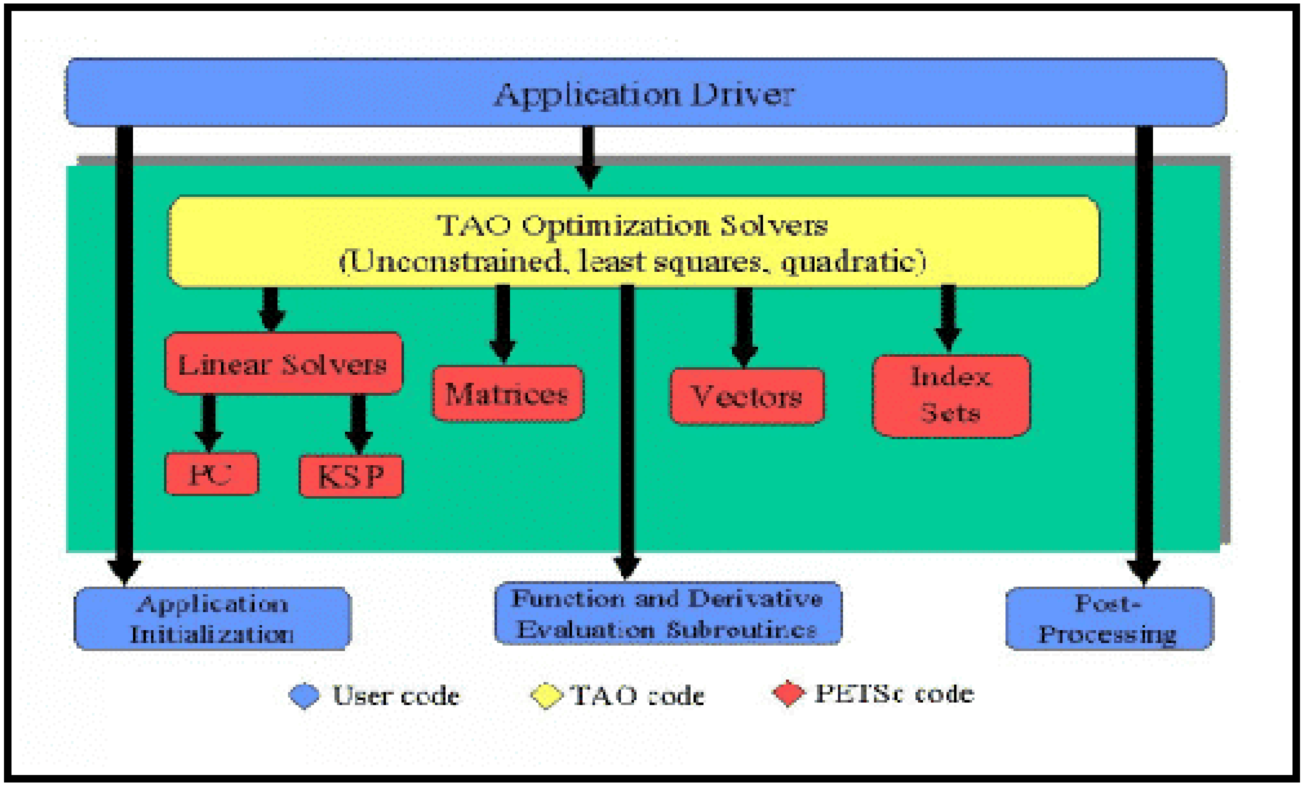
Classes of Problems in OPT++

- Four major classes of problems available
  - *NLF0(ndim, fcn, init_fcn, constraint)*
    - Basic nonlinear function, no derivative information available
  - *NLF1(ndim, fcn, init_fcn, constraint)*
    - Nonlinear function, first derivative information available
  - *FDNLF1(ndim, fcn, init_fcn, constraint)*
    - Nonlinear function, first derivative information approximated
  - *NLF2(ndim, fcn, init_fcn, constraint)*
    - Nonlinear function, first and second derivative information available

Classes of Solvers in OPT++

- Direct search
  - No derivative information required
- Conjugate Gradient
  - Derivative information may be available but doesn't use quadratic information
- Newton-type methods
  - Algorithm attempts to use/approximate quadratic information
  - Newton
  - Finite-Difference Newton
  - Quasi-Newton
  - NIPS

TAO - Interface with PETSc



TAO - Bound Constraint Optimization

- Conjugate Gradient
- Limited-Memory variable-metric algorithms
- Newton Algorithms where user supplies

The user must provide:

- The Function and its First Derivative
- For Newton Methods the second Derivative is also required (Hessian Matrix)

TAO - CG Algorithms

$$x_{k+1} = x_k + \alpha_k p_k$$

$$p_{k+1} = -\nabla f(x_k) + \beta_k p_k$$

where α_k is determined by a line search.

Three choices of β_k are possible ($g_k = \nabla f(x_k)$):

$$\beta_k^{FR} = \left(\frac{\|g_{k+1}\|}{\|g_k\|} \right)^2, \qquad \text{Fletcher-Reeves}$$

$$\beta_k^{PR} = \frac{\langle g_{k+1}, g_{k+1} - g_k \rangle}{\|g_k\|^2}, \qquad \text{Polak-Rivière}$$

$$\beta_k^{PR+} = \max \left\{ \beta_k^{PR}, 0 \right\}, \qquad \text{PR-plus}$$

TAO - Limited-Memory Variable

$$x_{k+1} = x_k - \alpha_k H_k \nabla f(x_k)$$

where α_k is determined by a line search.

The matrix H_k is defined in terms of information gathered during the previous m iterations.

- H_k is positive definite.
- Storage of H_k requires $2mn$ locations.
- Computation of $H_k \nabla f(x_k)$ costs $(8m + 1)n$ flops.

TAO - Trust Region Newton Algorithms

At each iteration the step s_k (approximately) minimizes

$$\min \{q_k(x_k + s) : s_i = 0,\ i \in \mathcal{A}_k,\ x_l \le x_k + s \le x_u,\ \|s\| \le \Delta_k\}$$

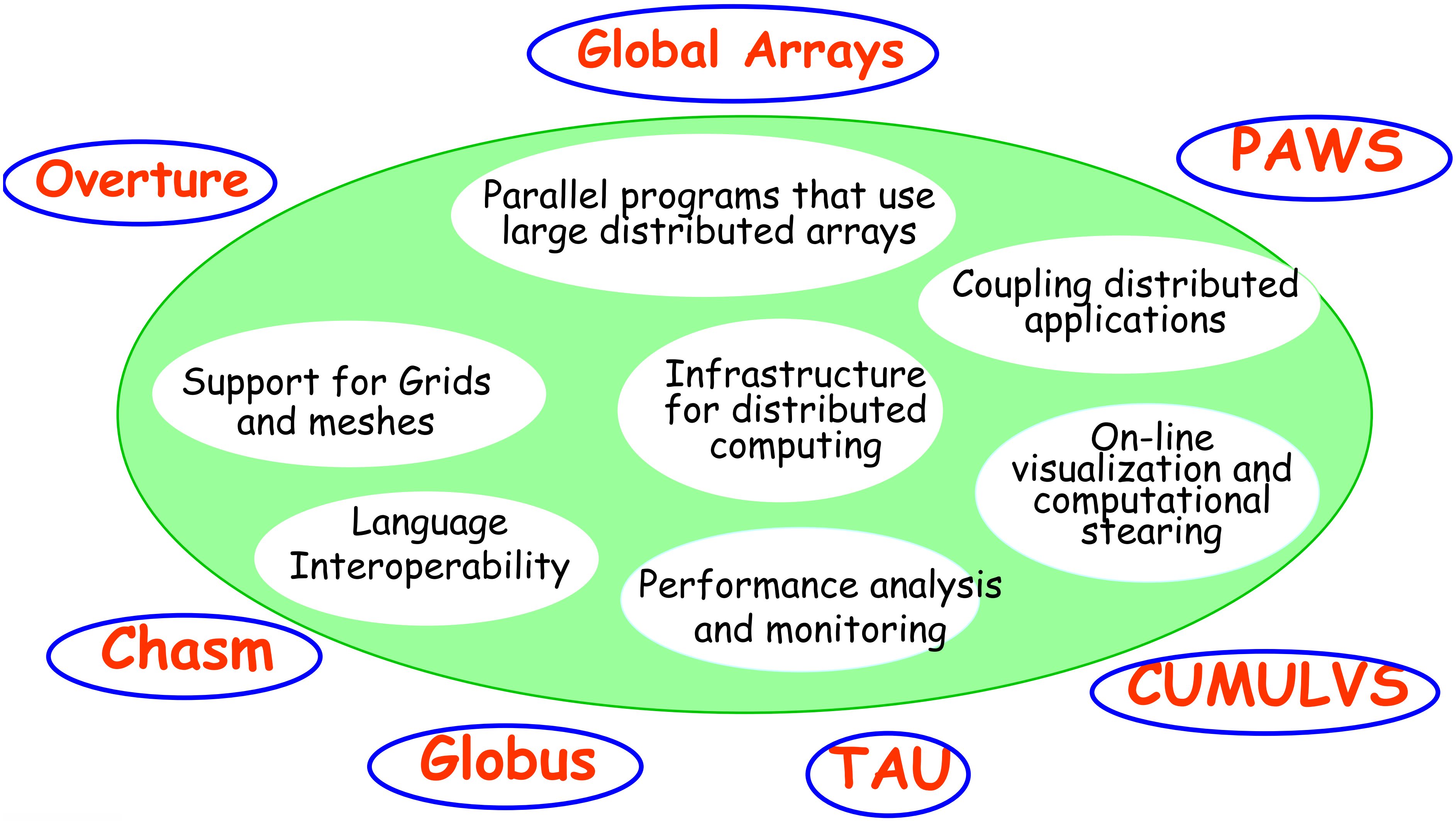
where q_k is the quadratic approximation,

$$q_k(w) = \langle \nabla f(x_k), w \rangle + \tfrac{1}{2}\langle w, \nabla^2 f(x_k) w \rangle,$$

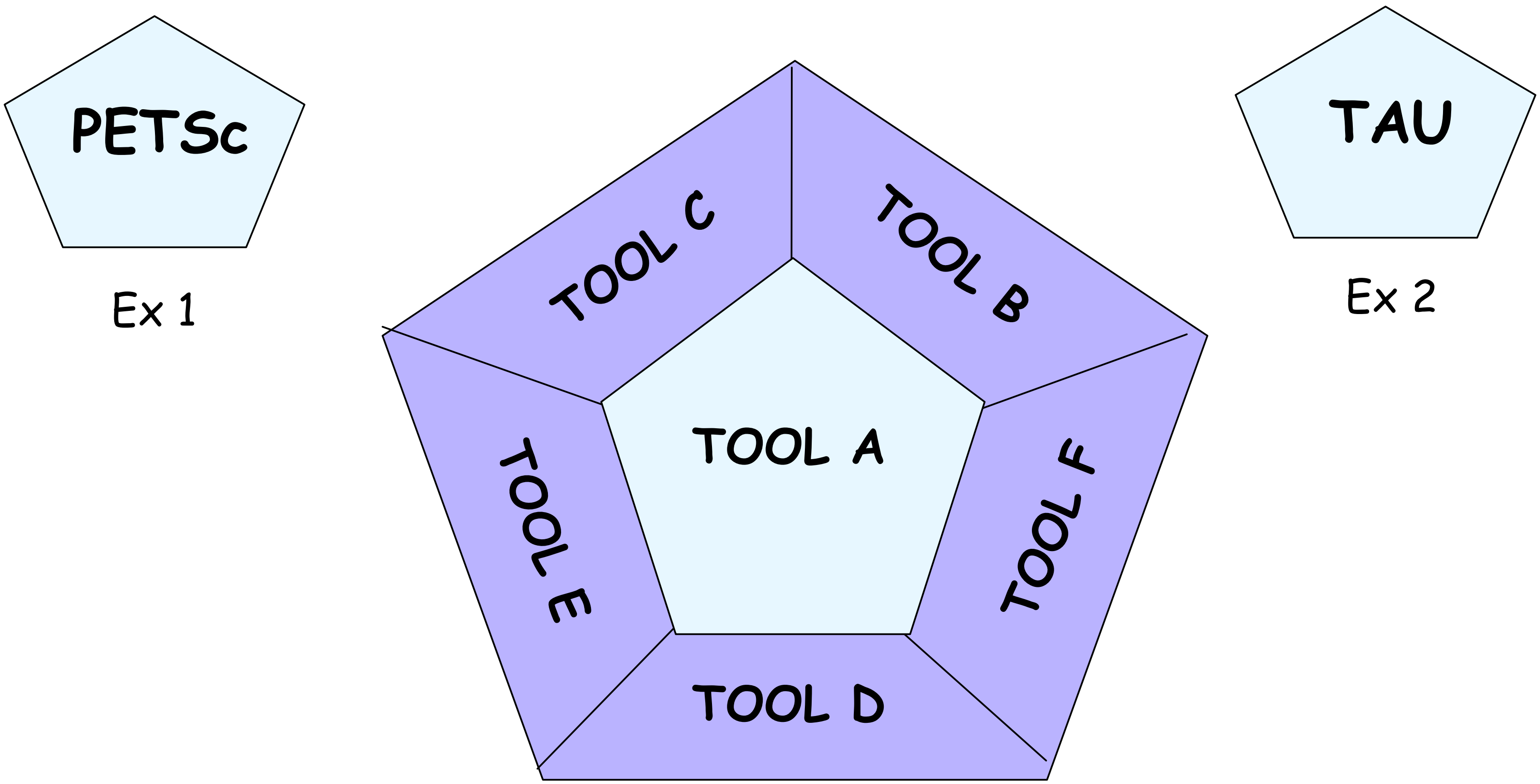
to the function, and Δ_k is the trust region bound.

- Predict an active set $\mathcal{A}_k$.
- Compute a step s_k
- $x_{k+1} = x_k + s_k$ if $f(x_k + s_k) < f(x_k)$, otherwise $x_{k+1} = x_k$.
- Update Δ_k.

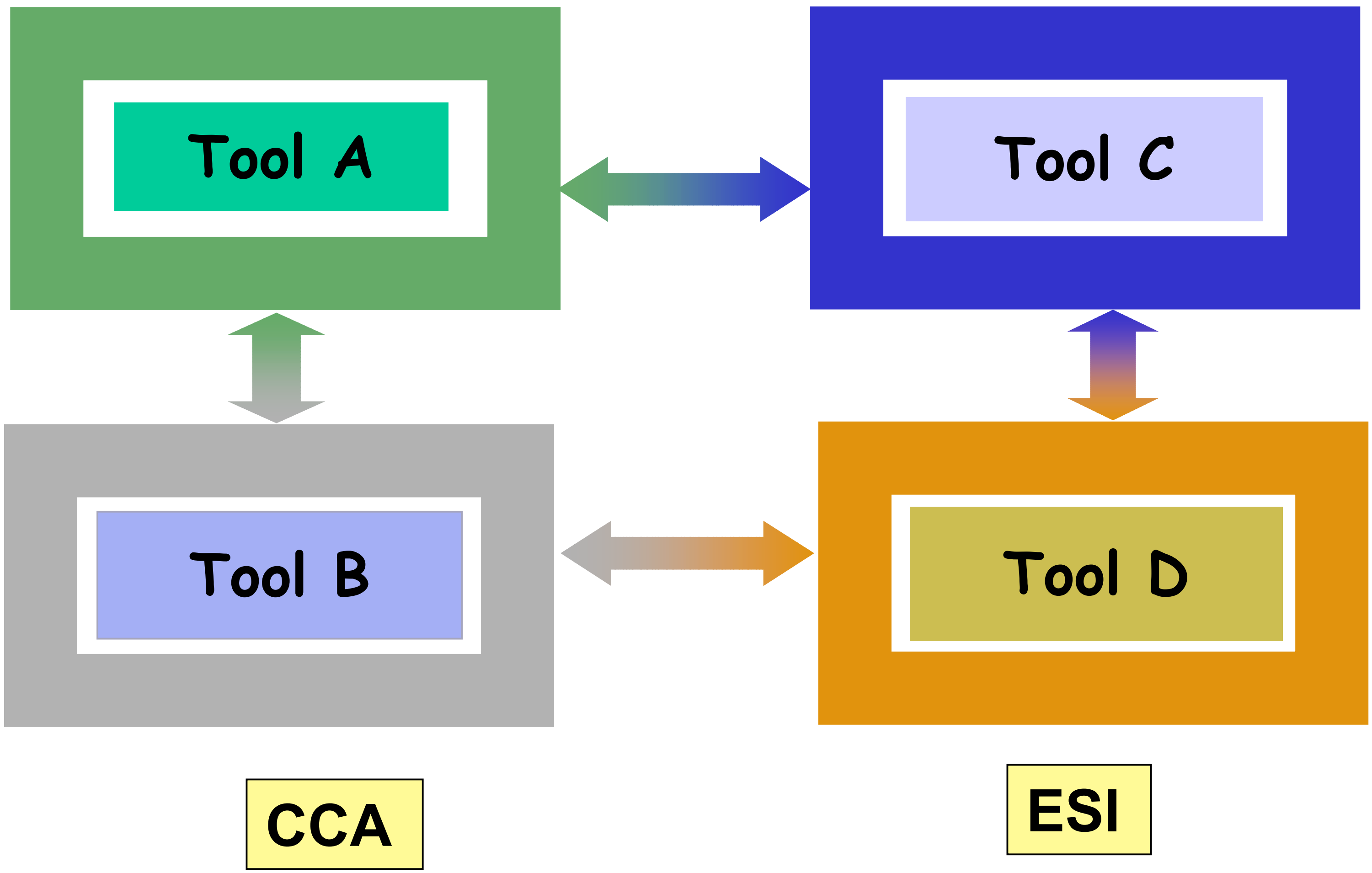
What codes are being developed?



Tool Interoperability
Tool-to-Tool



Component Technology!



PSE's and Frameworks

PMatlab

PyACTS

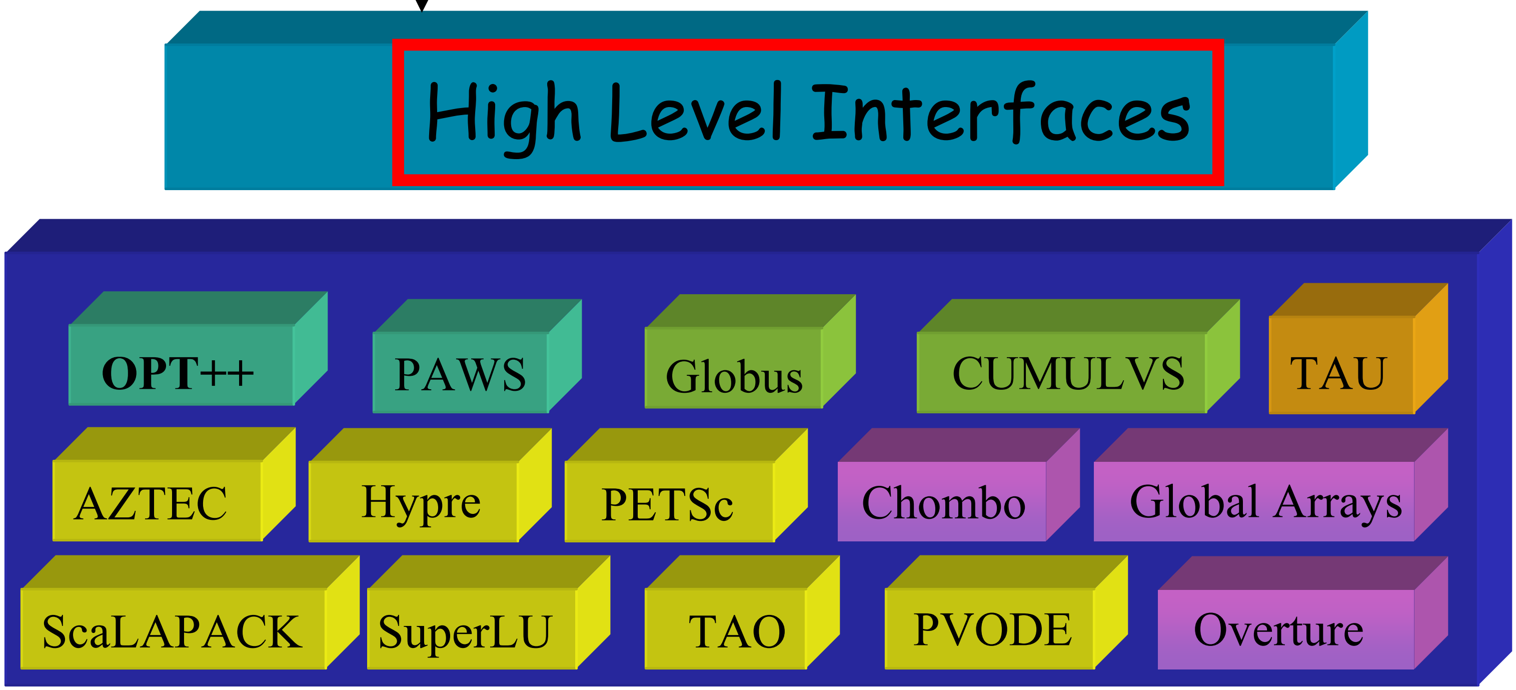
$$Ax = b$$

$$Az = \lambda z$$

View_field(T1)

$$A = U\Sigma V^T$$

User



PyACTS

Terminal

Window Edit Options Help

```
% run 4

-------------------------------------------------------------
 User: nkang                          Repo: mpccc
 Job Name: <none specified>           Group: mpccc
 Class Of Service: interactive        Job Class: interactive
 Job Accepted: Wed Jul 30 09:42:16 2003
-------------------------------------------------------------

llsubmit: Processed command file through Submit Filter: "/usr/common/nsg/etc/sub
filter".
>>>
import sys
>>>
sys.path.append('/u6/nkang/kn/pyacts_1/build/lib.aix-5.1-mpi-2.2')
>>>
import scalapack
>>>
scalapack.ex2("ex2_mat","ex2_rhs","sol",6,1,2,2,2,1)

Scalapack Example Program #2 (C-version) -- 07/24/2003
Solving AX=B
where A is a 6 by 6 matrix,
B is a 6 by 1 matrix,
with a block size of 2
Running on 4 processes, where the process grid is 2 by 2
INFO code returned by PDGESV = 0

According to the normalized residual the solution is correct.
||AX-B|| / (||X||*||A||*eps*N) =   1.25878215e-01

The solution is written to file sol

End of test.
>>>
```

More Tutorials

http://acts.nersc.gov/events/Workshop2004/

SIAM CSE05 Short Course + Workshop 2004 CD
Request: acts-support@nersc.gov - **It's FREE**!

Coming Soon! ACTS Workshop 2005
August 23-26, 2005